Using Azure Automation Accounts to Access Key Vaults

This is the second post in a series of blogs that focuses around Azure Automation. Check out “Exporting Azure RunAs Certificates for Persistence” for more info on how authentication works for Automation Accounts. In this installment, we’re going to focus on making use of Automation Accounts to gain access to sensitive data stored in Key Vaults.

High Level TLDR:

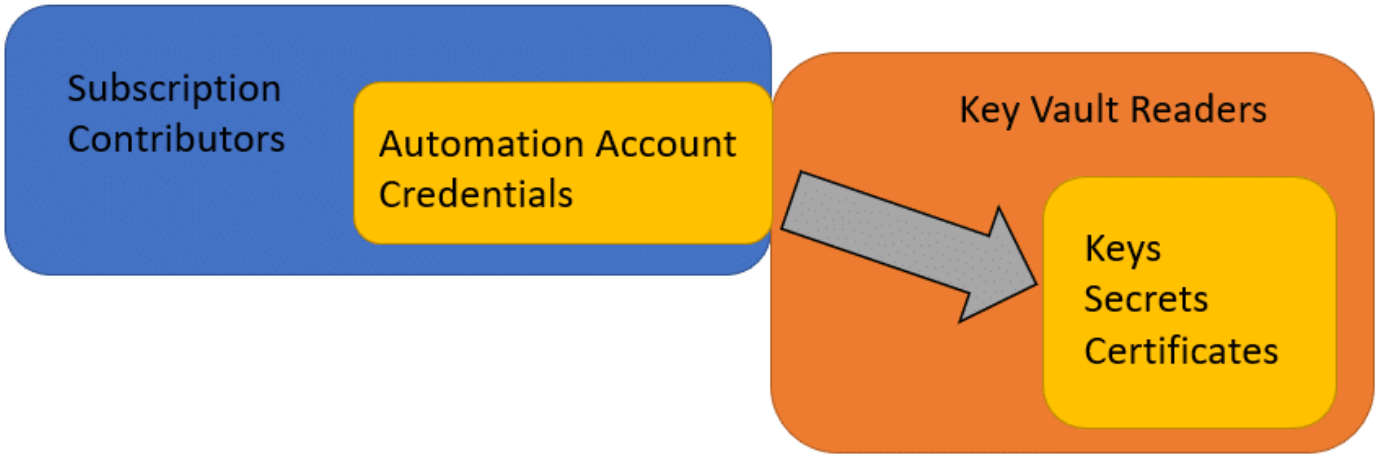
1. Gain access to an AzureAD account that has rights to create/modify/run automation runbooks in existing Automation Accounts
    - This AzureAD account doesn’t have access to any Key Vaults, but you want to read the vaults
2. Access an Automation Account configured with rights to read keys from a Key Vault
3. Add (or modify) a runbook for the Automation Account to access the Key Vault and dump the secrets
4. Use your newfound secrets for further pivoting in the environment

I have been frequently running into situations where I have contributor access to a subscription, but I’m unable to access the data in the Azure Key Vaults. Since I can’t grant myself access to the vaults (requires Owner rights), I’ve had to come up with some creative ways for accessing the Key Vaults with Contributor accounts.

Initial Access

Most of the time that we have access to a Contributor account in Azure, the account does not have access to any of the Key Vaults in the subscription. Security conscious developers/engineers will limit the rights for normal users and assign application specific accounts to handle accessing Key Vaults. This limits the liability put on user accounts, and keeps the secrets with the application service accounts.

While we may not have access to the Key Vaults, we do have contributor rights on the Automation Account runbooks. This grants us the rights to create/modify/run automation runbooks for the existing Automation Accounts, which allows us to run code as automation users, that may have rights to access Key Vaults.

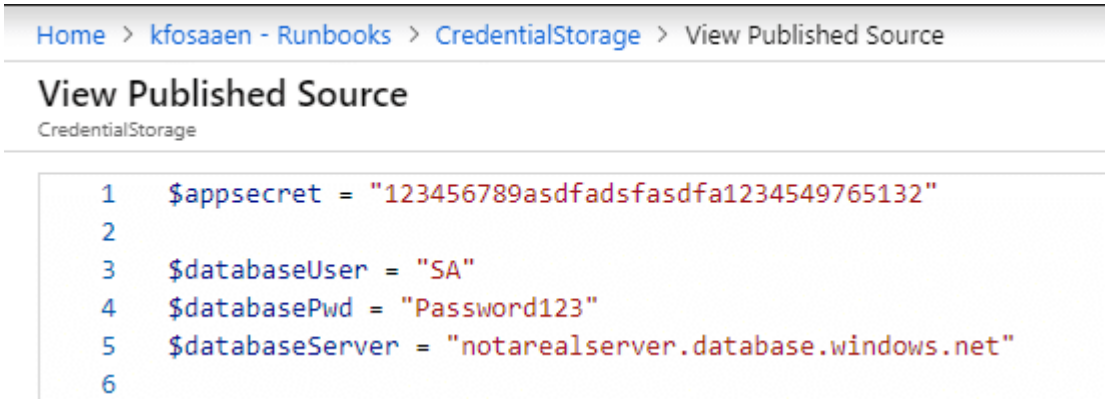


So why does this happen? As a best practice for automating specific tasks within Azure, engineers may vault keys/credentials that are used by automation runbooks. The Automation Accounts are then granted access to the Key Vaults to make use of the keys/credentials as part of the automation process to help abstract the credentials away from the runbook code and the users.

Common Automation Key Vault Applications:

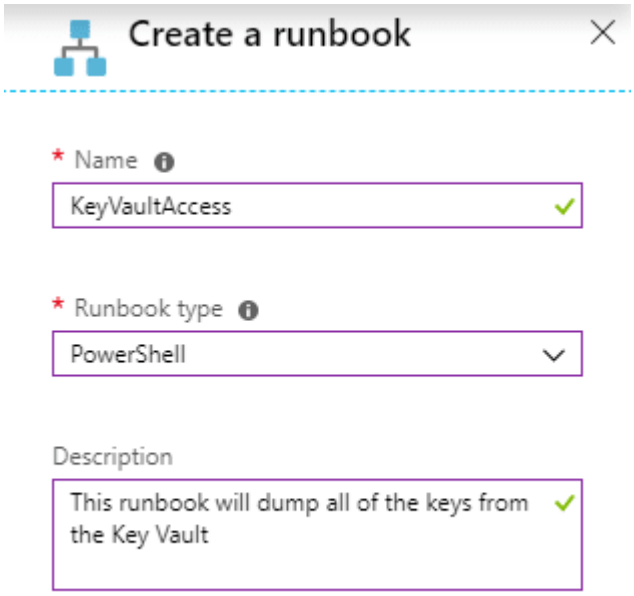
- Keys for encrypting data in an application
- Local administrator passwords for VMs
- SQL database credentials for accessing AzureSQL databases
- Access key storage for other Azure services

As a side note: Azure developers/engineers are getting better at making use of Key Vaults for automation credentials, but we still occasionally see credentials that are hard-coded in runbooks. If you have read access on runbooks, keep an eye out for hard-coded credentials.



Creating a New Runbook

In order to access the key vaults from the Automation Accounts, we will need to create a new runbook that will list out each of the vaults, and all of the keys for each vault. We will then run this runbook with the RunAs account, along with any credentials configured for the account. So far, this shotgun approach has been the easiest way to enumerate key values in a vault, but it's not the most opsec friendly method. In many of the environments that I've seen, there are specific alerting rules already set up for unauthorized access attempts to key vaults. So be careful when you're doing this.



It has been a little difficult trying to come up with a method for determining Automation user access before running a runbook in the Automation Account. There's no way to grab cleartext automation credentials from an Automation Account without running a runbook, and it's a little tricky (but possible) to get the Key Vault rights for RunAs accounts before running a runbook.

Grand scheme of things… you will need to run a runbook to pull the keys, so you might as well go for all the keys at once. If you want to be more careful with the Automation Accounts that you use for this attack, keep an eye out for runbooks that have code to specifically read from Key Vaults. Chances are good that the account has access to one or more vaults. You can also choose the specific automation accounts that you want to use in the following script.

Automating the Process

At a high level, here's what we will accomplish with the "Get-AzureKeyVaults-Automation" PowerShell function":

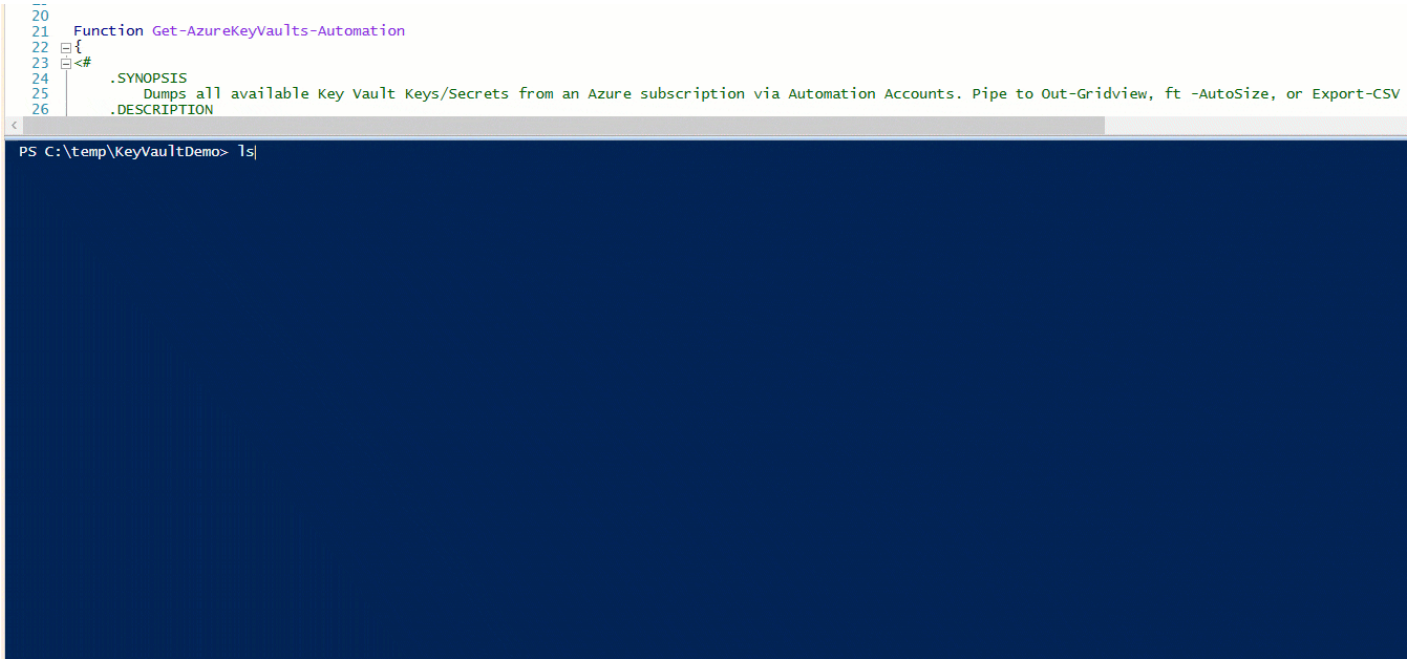
1. List the Automation Accounts
    - Select the Automation Accounts that you want to use
2. Iterate through the list and run a standardized runbook in each selected account (with a randomized job name)
    - List all of the Key Vaults
    - Attempt to read every key with the current account
    - Complete these actions with both the RunAs and Stored Credential accounts
3. Output all of the keys that you can access
    - There may be duplicate results at the end due to key access overlap

This PowerShell function is available under the MicroBurst repository. You can find MicroBurst here – https://github.com/NetSPI/MicroBurst

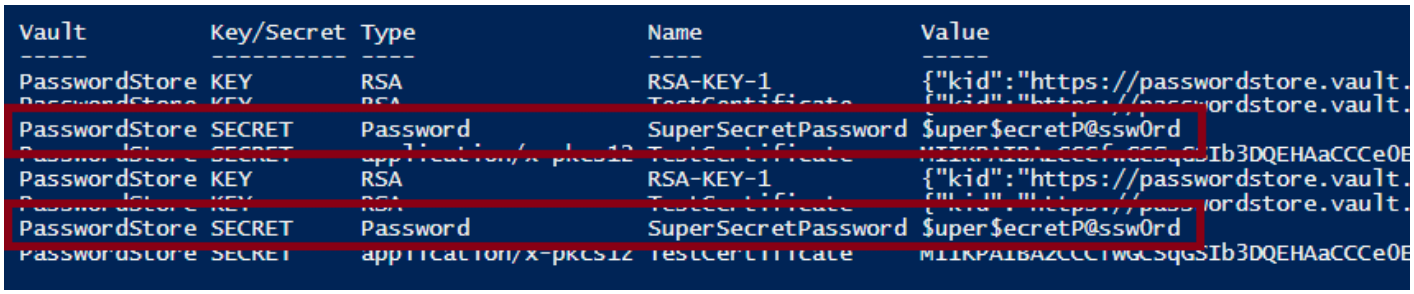
Example

Here's a sample run of the function in my test domain:

```
Get-AzureKeyVaults-Automation -ExportCerts Y -Subscription "SUBSCRIPTION_NAME"
-Verbose | ft -AutoSize
```



Example Output:



Conclusions

For the Attackers – You may have a situation where you need to access Key Vaults with a lesser privileged user. Hopefully the code/function presented in this blog allows you to move laterally to read the secrets in the vault.

For the Defenders – If you're using Automation Accounts in your subscription, there's a good chance that you will need to configure an Automation Account with Key Vault reader rights. When doing this, make sure that you're limiting the Key Vaults that the account has access to. Additionally, be careful with who you give subscription contributor access to. If a contributor is compromised, your Automation Accounts may just give up your secrets.

Update – 12/30/2019

This issue was not initially reported to MSRC, due to the fact that it’s a user misconfiguration issue and not eligible for reporting per the MSRC guidelines (“Security misconfiguration of a service by a user“). However, they became aware of the blog and ended up issuing a CVE for it – https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0962